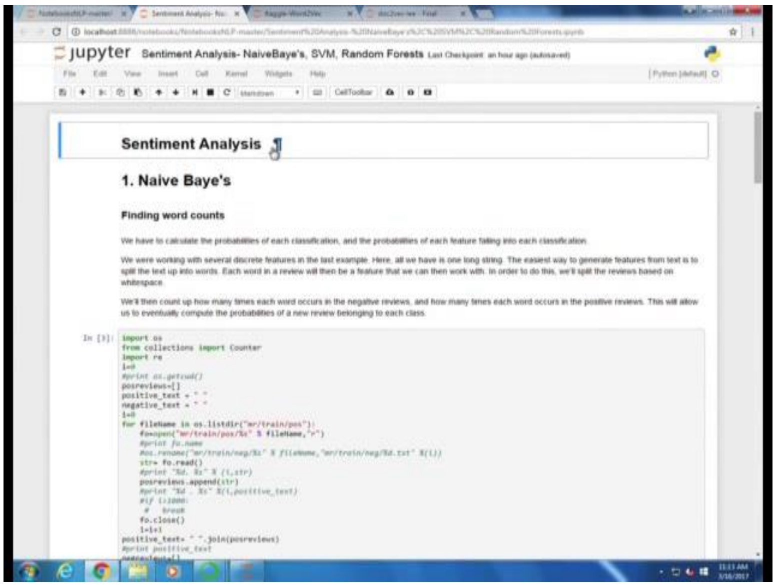
Natural Language Processing

Prof. Amrith Krishna
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 59
Tutorial

Hello everyone, welcome to the tutorial section for the NLP course. So, today we will be looking into how to use or how to build a classifier for one of the most popular task in NLP sentiment analysis and we will be looking to how we create features and how we can build a supervised machine learning classifier for the purpose of sentiment analysis.

(Refer Slide Time: 00:24)


jupyter
Sentiment Analysis- NaiveBaye's, SVM, Random Forests Last Checkpoint an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python [default]
CellToolbar

Sentiment Analysis

1. Naive Baye's

Finding word counts

We have to calculate the probabilities of each classification, and the probabilities of each feature falling into each classification.

We were working with several discrete features in the last example. Here, all we have is one long string. The easiest way to generate features from text is to split the text up into words. Each word in a review will then be a feature that we can then work with. In order to do this, we'll split the reviews based on whitespace.

We'll then count up how many times each word occurs in the negative reviews, and how many times each word occurs in the positive reviews. This will allow us to eventually compute the probabilities of a new review belonging to each class.

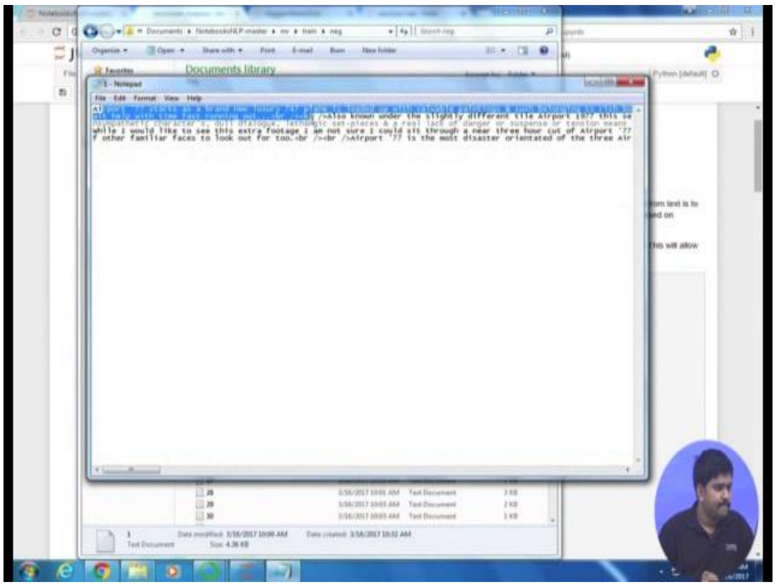
```
In [3]: import os
        from collections import Counter
        import re
        i=0
        #print os.getcwd()
        posreviews=[]
        positive_text = " "
        negative_text = " "
        i=0
        for fileName in os.listdir("mr/train/pos"):
            fo=open("mr/train/pos/%s" % fileName,"r")
            #print fo.name
            #os.rename("mr/train/neg/%s" % fileName,"mr/train/neg/%d.txt" %(i))
            str= fo.read()
            #print "%d. %s" % (i,str)
            posreviews.append(str)
            #print "%d . %s" %(i,positive_text)
            #if i>1000:
             #   break
            fo.close()
            i=i+1
        positive_text= " ".join(posreviews)
        #print positive_text
```

So, we will essentially look into how we can build a Naive Bayes classifier and after that we will just give you pointers towards how to build other classifiers. The notebook provided to you will have code snippets for building other type of classifiers, I will not be discussing it here.

So, sentiment analysis is essentially a task where you are given a document which can be a sentence or a set of sentences and you are given with and you have to label it whether it belongs to a positive sentiment or a negative sentiment. This is the most simplistic representation or a task that we can think of sentiment analysis. Of course, as per the demands, this complexity of the task can be increased or like depending on the requirement from the customer or the user, there might be other levels of sentiment that might be applied here. So, here what we will be doing is that we will be taking about 5000 documents which are 5000 movie reviews and we see and we are already provided with the labels for those 5000 documents.

Here we can find about 2500 negative reviews and positive reviews about different films.

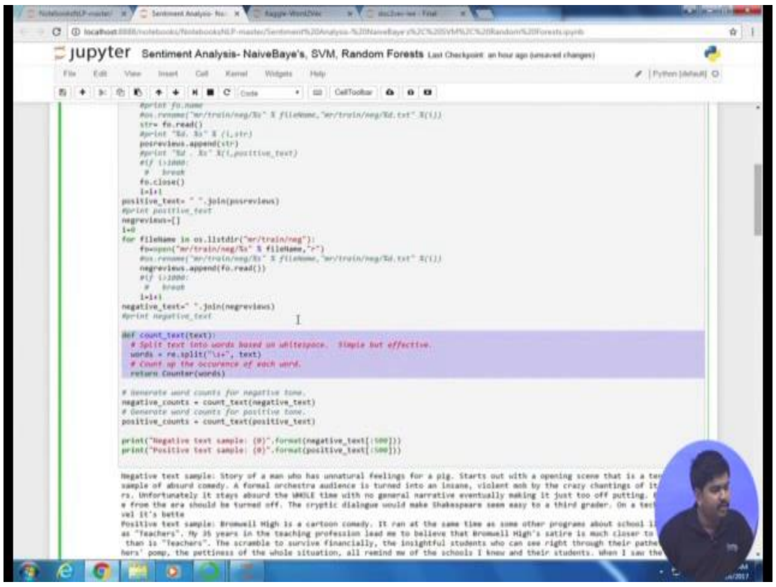
(Refer Slide Time: 02:05)



```
Airport '77 starts as a brand new luxury 747 plane is loaded up with [illegible]
[illegible] help with time fast running out...<br /><br />Also known under the slightly different tile Airport 1977 this se
nsympathetic character's, dull dialogue, lethargic set-pieces & a real lack of danger or suspense or tension means
while i would like to see this extra footage i am not sure i could sit through a near three hour cut of Airport '77
f other familiar faces to look out for too.<br /><br />Airport '77 is the most disaster orientated of the three Air
```

Now once this are provided, what we have to do is that we have to build a supervise classifier where we tag the positives as 1 and the negatives as 0 and then we build the classifier. So, here each document is your input and your models have to predict the sentiment of a new document when provided in the similar representation. So, we have to first convert each of those documents which is a training document in your vectorial representation. So, for that purpose, what we do here is that we take each positive and negative documents into separate data structure which is basically a list.

(Refer Slide Time: 03:28)



Sentiment Analysis- NaiveBaye's, SVM, Random Forests

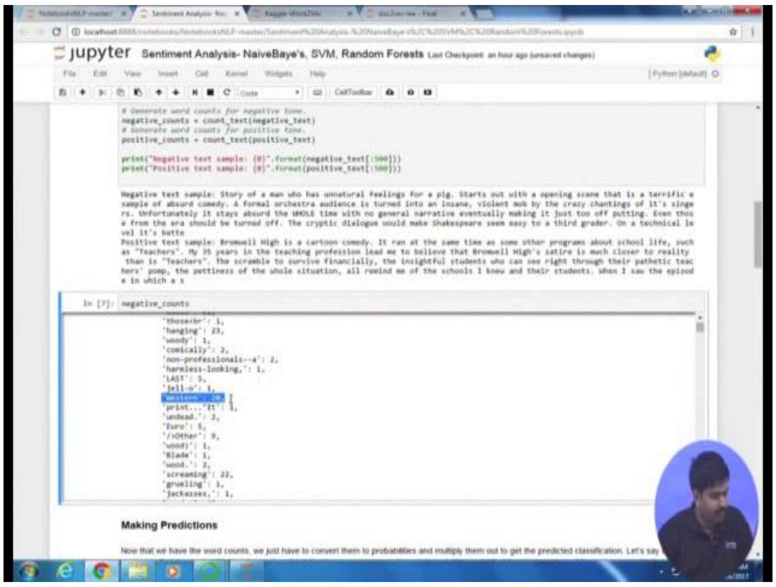
```
    #print fo.name
    #os.rename("mr/train/neg/%s" % fileName,"mr/train/neg/%d.txt" %(i))
    str= fo.read()
    #print "%d. %s" % (i,str)
    posreviews.append(str)
    #print "%d . %s" %(i,positive_text)
    #if i>1000:
     #   break
    fo.close()
    i=i+1
positive_text= " ".join(posreviews)
#print positive_text
negreviews=[]
i=0
for fileName in os.listdir("mr/train/neg"):
    fo=open("mr/train/neg/%s" % fileName,"r")
    #os.rename("mr/train/neg/%s" % fileName,"mr/train/neg/%d.txt" %(i))
    negreviews.append(fo.read())
    #if i>1000:
     #   break
    i=i+1
negative_text=" ".join(negreviews)
#print negative_text

def count_text(text):
  # Split text into words based on whitespace.  Simple but effective.
  words = re.split("\s+", text)
  # Count up the occurence of each word.
  return Counter(words)

# Generate word counts for negative tone.
negative_counts = count_text(negative_text)
# Generate word counts for positive tone.
positive_counts = count_text(positive_text)

print("Negative text sample: {0}".format(negative_text[:500]))
print("Positive text sample: {0}".format(positive_text[:500]))
```

```
Negative text sample: Story of a man who has unnatural feelings for a pig. Starts out with a opening scene that is a te
sample of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it
rs. Unfortunately it stays absurd the WHOLE time with no general narrative eventually making it just too off putting.
e from the era should be turned off. The cryptic dialogue would make Shakespeare seem easy to a third grader. On a tec
vel it's bette
Positive text sample: Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school l
as "Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High's satire is much closer to
 than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathe
hers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the
```

And once we do that what we do is that we count the number of individual words that is appearing in each of these sets.

We can find that since each document here is given as a separate file, we use the inbuilt OS library to traverse through each of the file given in the folder pos or the positive and then we append the content to a particular list. Once we do that we define a function called count text which basically looks for individual words in each of the document and also it keeps a track of the count or the number of time it is appearing.

(Refer Slide Time: 04:39)



Sentiment Analysis- NaiveBaye's, SVM, Random Forests

```
# Generate word counts for negative tone.
negative_counts = count_text(negative_text)
# Generate word counts for positive tone.
positive_counts = count_text(positive_text)

print("Negative text sample: {0}".format(negative_text[:500]))
print("Positive text sample: {0}".format(positive_text[:500]))
```

```
Negative text sample: Story of a man who has unnatural feelings for a pig. Starts out with a opening scene that is a terrific e
xample of absurd comedy. A formal orchestra audience is turned into an insane, violent mob by the crazy chantings of it's singe
rs. Unfortunately it stays absurd the WHOLE time with no general narrative eventually making it just too off putting. Even thos
e from the era should be turned off. The cryptic dialogue would make Shakespeare seem easy to a third grader. On a technical le
vel it's bette
Positive text sample: Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life, such
as "Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High's satire is much closer to reality
 than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathetic teac
hers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the episod
e in which a s
```

```
In [7]: negative_counts
```

```
 'those<br': 1,
 'hanging': 23,
 'woody': 1,
 'comically': 2,
 'non-professionals--a': 2,
 'harmless-looking,': 1,
 'LAST': 5,
 'Jell-o': 1,
 'western': 20,
 'print..."It': 1,
 'undead.': 3,
 'Euro': 5,
 '/>Other': 9,
 'wood)': 1,
 'Blade': 1,
 'wood.': 2,
 'screaming': 22,
 'grueling': 1,
 'jackasses,': 1,
```

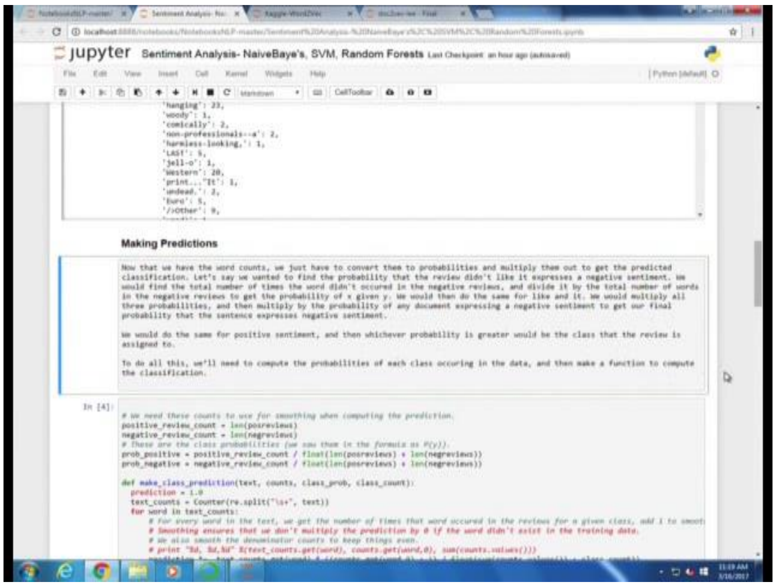
Making Predictions

Now that we have the word counts, we just have to convert them to probabilities and multiply them out to get the predicted classification. Let's say

We can just have a look of how the variable negative counts looks like, if you see here for negative the words pineapple, consider, daring, etcetera are appearing in different counts. So, we can find western appearing 20 times, screening appear 22 times, wooden appears 42 times and so on. So, here we have not done any pre processing for as it is so, far illustrative purposes. So, in this corpus may be the daring might be at separate word than this word because it has some extra symbols in it. These all need to be taken care when we go for a practical application. We do not do any case folding or any removal of other special symbols, but all these needs to be taken care.

So, once we have this separate dictionaries, now what we do is do here is that we just find out, how often a word is appearing in both positive and negative context. It is often possible that some of these words can appear in positive and negative context. So, the very basic idea here is that if a word appears more or less like equally in both the sets, they does not have much value, if a word words presents in one of these sets say positive context is queued with compared to what is happening with the negative set then that is more likely a positive sentiment word. If a word is more likely to belong to a negative set it is more likely a negative sentiment word, if it is more likely to belong to a positive set, it is a positive word.

(Refer Slide Time: 06:26)


Sentiment Analysis- NaiveBaye's, SVM, Random Forests

```
              'hanging': 23,
              'woody': 1,
              'comically': 2,
              'non-professionals--a': 2,
              'harmless-looking,': 1,
              'LAST': 5,
              'jell-o': 1,
              'Western': 20,
              'print..."It': 1,
              'undead.': 2,
              'Euro': 5,
              '/>Other': 9,
```

Making Predictions

Now that we have the word counts, we just have to convert them to probabilities and multiply them out to get the predicted classification. Let's say we wanted to find the probability that the review didn't like it expresses a negative sentiment. We would find the total number of times the word didn't occured in the negative reviews, and divide it by the total number of words in the negative reviews to get the probability of x given y. We would then do the same for like and it. We would multiply all three probabilities, and then multiply by the probability of any document expressing a negative sentiment to get our final probability that the sentence expresses negative sentiment.

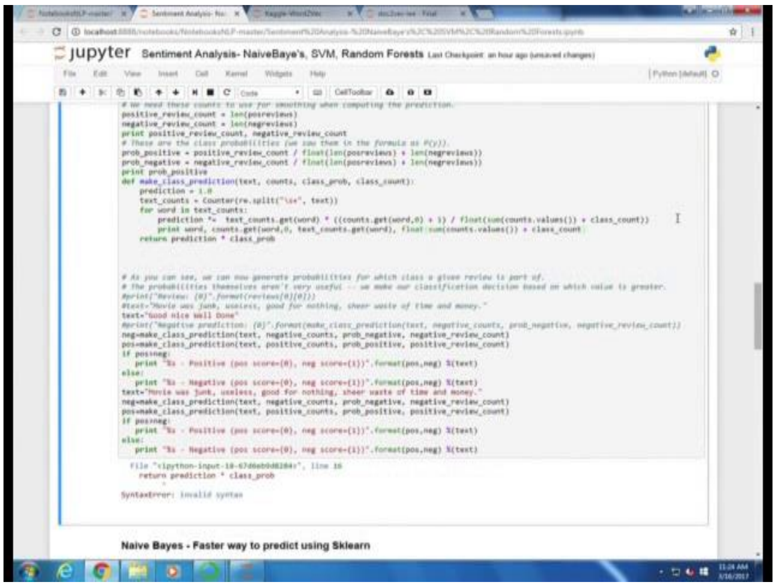
We would do the same for positive sentiment, and then whichever probability is greater would be the class that the review is assigned to.

To do all this, we'll need to compute the probabilities of each class occuring in the data, and then make a function to compute the classification.

```
In [4]:
# We need these counts to use for smoothing when computing the prediction.
positive_review_count = len(posreviews)
negative_review_count = len(negreviews)
# These are the class probabilities (we saw them in the formula as P(y)).
prob_positive = positive_review_count / float(len(posreviews) + len(negreviews))
prob_negative = negative_review_count / float(len(posreviews) + len(negreviews))

def make_class_prediction(text, counts, class_prob, class_count):
  prediction = 1.0
  text_counts = Counter(re.split("\s+", text))
  for word in text_counts:
      # For every word in the text, we get the number of times that word occured in the reviews for a given class, add 1 to smoot
      # Smoothing ensures that we don't multiply the prediction by 0 if the word didn't exist in the training data.
      # We also smooth the denominator counts to keep things even.
      # print "%d, %d,%d" %(text_counts.get(word), counts.get(word,0), sum(counts.values()))
```

We have the description here where we what we do is that we would find the total number of times a particular word that occur in the corpus and we just find the probability of it with its total count.

(Refer Slide Time: 06:50)


Sentiment Analysis- NaiveBaye's, SVM, Random Forests

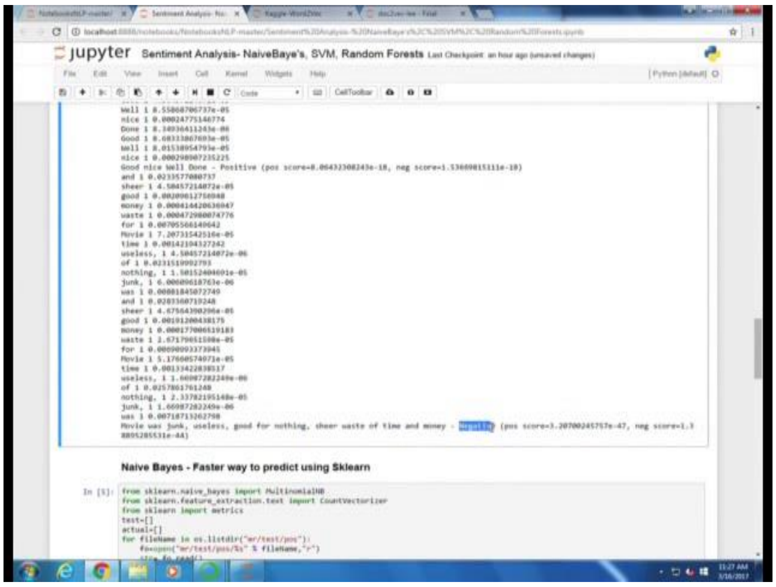
```
# We need these counts to use for smoothing when computing the prediction.
positive_review_count = len(posreviews)
negative_review_count = len(negreviews)
print positive_review_count, negative_review_count
# These are the class probabilities (we saw them in the formula as P(y)).
prob_positive = positive_review_count / float(len(posreviews) + len(negreviews))
prob_negative = negative_review_count / float(len(posreviews) + len(negreviews))
print prob_positive
def make_class_prediction(text, counts, class_prob, class_count):
    prediction = 1.0
    text_counts = Counter(re.split("\s+", text))
    for word in text_counts:
        prediction *=  text_counts.get(word) * ((counts.get(word,0) + 1) / float(sum(counts.values()) + class_count))
        print word, counts.get(word,0, text_counts.get(word), float sum(counts.values()) + class_count
    return prediction * class_prob



# As you can see, we can now generate probabilities for which class a given review is part of.
# The probabilities themselves aren't very useful -- we make our classification decision based on which value is greater.
#print("Review: {0}".format(reviews[0][0]))
#text="Movie was junk, useless, good for nothing, sheer waste of time and money."
text="Good nice Well Done"
#print("Negative prediction: {0}".format(make_class_prediction(text, negative_counts, prob_negative, negative_review_count))
neg=make_class_prediction(text, negative_counts, prob_negative, negative_review_count)
pos=make_class_prediction(text, positive_counts, prob_positive, positive_review_count)
if pos>neg:
   print "%s - Positive (pos score={0}, neg score={1})".format(pos,neg) %(text)
else:
   print "%s - Negative (pos score={0}, neg score={1})".format(pos,neg) %(text)
text="Movie was junk, useless, good for nothing, sheer waste of time and money."
neg=make_class_prediction(text, negative_counts, prob_negative, negative_review_count)
pos=make_class_prediction(text, positive_counts, prob_positive, positive_review_count)
if pos>neg:
   print "%s - Positive (pos score={0}, neg score={1})".format(pos,neg) %(text)
else:
   print "%s - Negative (pos score={0}, neg score={1})".format(pos,neg) %(text)

  File "<ipython-input-18-67d6eb0d8204>", line 16
    return prediction * class_prob
         ^
SyntaxError: invalid syntax
```

Naive Bayes - Faster way to predict using Sklearn

If we see what exactly is prob positive. So, what we do here is that we took to represent the notion of cuteness or more likelihood of something being occurring in one set than the other, we convert them to a suitable probability distribution and what we find here is that the positive review count and negative review count. They are more or less equal they are like roughly 2500 a little bit here and there, but roughly they are equal. So, yes we can find what is the respective count.

We can find there is a slight difference of 300 also between both of them, but we essentially have more or less 2500 documents in each of those classes. Now so we can roughly say that they are like near to 0.5 in probability each, there is the number of documents that has prob positive sentiment and negative sentiment and are roughly half once we have that what remains is to find the individual probability of each word that is present. So, in order to do that, let us see, how this work, how this is calculated? So, print word and we will also find count, it seems to have a minor error in our work. So, indentation is something that is pretty important in python.

(Refer Slide Time: 11:41)



jupyter Sentiment Analysis- NaiveBaye's, SVM, Random Forests Last Checkpoint: an hour ago (unsaved changes)

```
Well 1 8.55868706737e-05
nice 1 0.00024775146774
Done 1 8.10936411243e-06
Good 1 8.68333867693e-05
Well 1 8.01538954793e-05
nice 1 0.000298907235225
Good nice Well Done - Positive (pos score=8.06432308243e-18, neg score=1.53669815111e-18)
and 1 0.0233577080737
sheer 1 4.50457214072e-05
good 1 0.00209612756048
money 1 0.000414420636047
waste 1 0.000472980074776
for 1 0.00705566140642
Movie 1 7.20731542516e-05
time 1 0.00142104327242
useless, 1 4.50457214072e-06
of 1 0.0231519992793
nothing, 1 1.50152404601e-05
junk, 1 6.00609618763e-06
was 1 0.00081845072749
and 1 0.0283360719248
sheer 1 4.67564390296e-05
good 1 0.00191200438175
money 1 0.000177006519183
waste 1 2.67179051508e-05
for 1 0.00690093373945
Movie 1 5.17600574071e-05
time 1 0.00133422838517
useless, 1 1.66987282249e-06
of 1 0.0257861761248
nothing, 1 2.33782195148e-05
junk, 1 1.66987282249e-06
was 1 0.00718713262798
Movie was junk, useless, good for nothing, sheer waste of time and money - Negative (pos score=3.20700245757e-47, neg score=1.3
8895285531e-44)
```

Naive Bayes - Faster way to predict using Sklearn

```
In [5]: from sklearn.naive_bayes import MultinomialNB
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn import metrics
        test=[]
        actual=[]
        for fileName in os.listdir("mr/test/pos"):
            fo=open("mr/test/pos/%s" % fileName,"r")
```

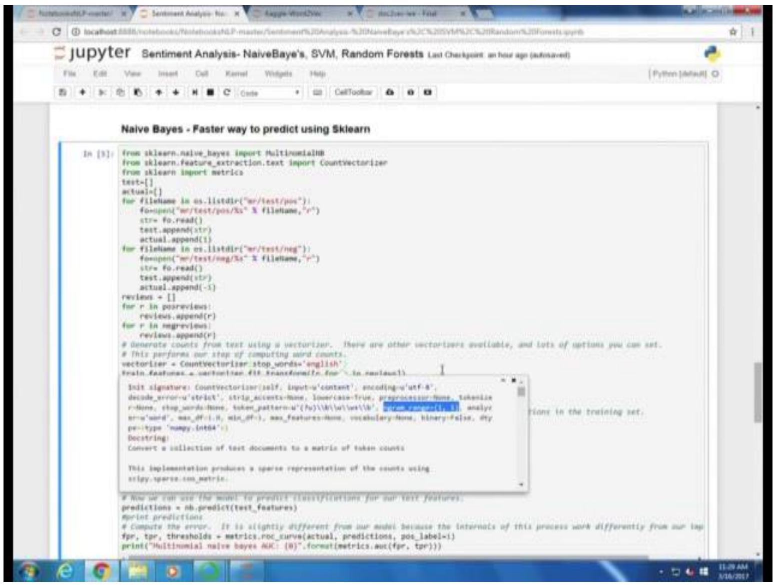
Here what we can find is that we can find the probability of each word belonging to a particular class. So, the function is defined in such a way that we first calculate all the prob, all the probability, all the words probability of belonging to the negative class then we find the probability of each of the word in the same document to belong to the positive class. So, suppose our new sentence is good, nice, well and done, we will calculate first probability of those words belonging to the negative counts and for the negative probability set and then to the positive probability set.

If we see here, so done, good, well and nice, first when they are given to the negatives, we are getting a probability of 3.003 into 10 to the power of minus 6 as compared to 8.34 into 10 to the power minus 6, similarly for other values also like good negative has a probability of appearing with 4.5 into 10 to the power minus 5 where is to good in positive is higher than that. So, once we have this individual scores it take the product of this course as shown here and we find that the positive score for this document which convince this 4 words is 8.06 into the power 10 power minus 8 as compared to the negative score which is much lesser than this. So, we can say that overall sentiment for this review is positive.

Now let us feed for another sentence where the sentence is the movie was junk useless good for nothing sheer waste of time and money. So, it does not take much to think that what will be the label for this document, but what we have to see is that whether the system is able to capture that see if you find the sentiment of this particular document is negative and we can find that this particular document as a very huge difference in its positive score and negative score. We can find that the value here is in the order of 10 to the power minus 44 while the positive score is somewhere at in the value of 10 to the power of minus 47.

So, this means this particular document is labeled as negative and the system is able to capture both the cases of positive and negative documents are released for these 2 test document cases. So, this is how we internally calculate the values.

(Refer Slide Time: 15:01)



Sentiment Analysis- NaiveBaye's, SVM, Random Forests

Naive Bayes - Faster way to predict using Sklearn

```
In [3]: from sklearn.naive_bayes import MultinomialNB
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn import metrics
        test=[]
        actual=[]
        for fileName in os.listdir("mr/test/pos"):
            fo=open("mr/test/pos/%s" % fileName,"r")
            str= fo.read()
            test.append(str)
            actual.append(1)
        for fileName in os.listdir("mr/test/neg"):
            fo=open("mr/test/neg/%s" % fileName,"r")
            str= fo.read()
            test.append(str)
            actual.append(-1)
        reviews = []
        for r in posreviews:
            reviews.append(r)
        for r in negreviews:
            reviews.append(r)
        # Generate counts from text using a vectorizer.  There are other vectorizers available, and lots of options you can set.
        # This performs our step of computing word counts.
        vectorizer = CountVectorizer(stop_words='english')
        train_features = vectorizer.fit_transform([r for r in reviews])
```

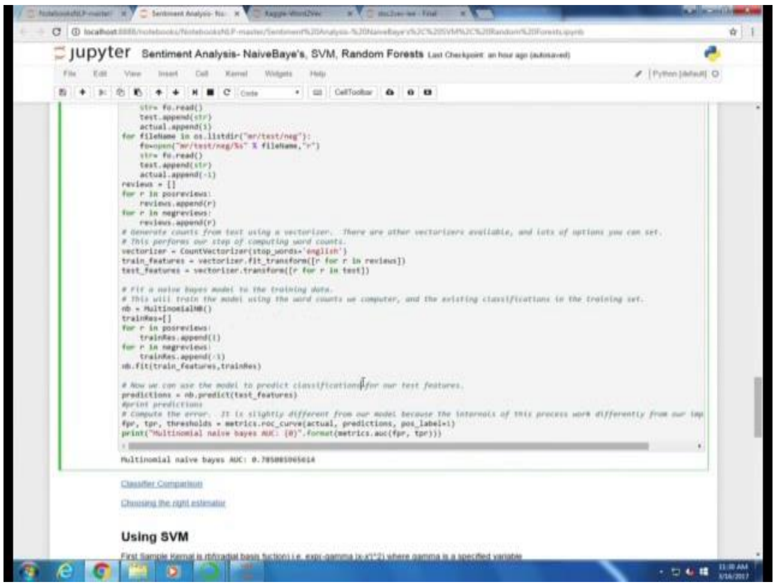
```
Init signature: CountVectorizer(self, input=u'content', encoding=u'utf-8',
decode_error=u'strict', strip_accents=None, lowercase=True, preprocessor=None, tokenize
r=None, stop_words=None, token_pattern=u'(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyz
er=u'word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dty
pe=<type 'numpy.int64'>)
Docstring:
Convert a collection of text documents to a matrix of token counts

This implementation produces a sparse representation of the counts using
scipy.sparse.coo_matrix.
```

```
        # Now we can use the model to predict classifications for our test features.
        predictions = nb.predict(test_features)
        #print predictions
        # Compute the error.  It is slightly different from our model because the internals of this process work differently from our imp
        fpr, tpr, thresholds = metrics.roc_curve(actual, predictions, pos_label=1)
        print("Multinomial naive bayes AUC: {0}".format(metrics.auc(fpr, tpr)))
```

Now, how we can use a Naive Bayes; Naive Bayes classifier for a production ready purpose. So, here what we do is that we use the Scikit learn library they have a Naive Bayes classifier already implemented what we will do is that we will use this classifier to do the same task that is shown here, but we test it on a larger number of test data. So, while giving he input to the system we have to again convert it into a suitable vectorial representation that this particular library understands. So, we use the count vector is a function which basically represents a document in terms of the number of times a particular word is appearing in that document and it by default removes the stop words or it has other possible functions as well.

This particular function provides a whole range of facilities that makes our job much easier like we can often use different multi word patterns for multi word ranges like for example, if you have a word like hot dog though they are essentially separated by space, hot dog does not, when we take the individual words hot or dog, it does not capture the notion that is represented by hot dog. So, it is assumed to be a multi word expression. So, we will be; so these kind of Engram range functionality will help you to capture those multi word expressions and of course, we have to use some filtering criteria to remove the unnecessary ones and we convert both the training vectors and the test vectors into this vectorial representation. So, there is again a small difference in the function called that we are doing fit transform and transform. So, fit and transform are actually two different methods and this Scikit learn function fit transform convert basically performs both the functions together. So, in fit we have to first build the vocabulary or the unique word.

(Refer Slide Time: 16:48)


jupyter Sentiment Analysis- NaiveBaye's, SVM, Random Forests Last Checkpoint: an hour ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python [default]
Code
CellToolbar

```
    str= fo.read()
    test.append(str)
    actual.append(1)
for fileName in os.listdir("mr/test/neg"):
    fo=open("mr/test/neg/%s" % fileName,"r")
    str= fo.read()
    test.append(str)
    actual.append(-1)
reviews = []
for r in posreviews:
    reviews.append(r)
for r in negreviews:
    reviews.append(r)
# Generate counts from text using a vectorizer.  There are other vectorizers available, and lots of options you can set.
# This performs our step of computing word counts.
vectorizer = CountVectorizer(stop_words='english')
train_features = vectorizer.fit_transform([r for r in reviews])
test_features = vectorizer.transform([r for r in test])

# Fit a naive bayes model to the training data.
# This will train the model using the word counts we computer, and the existing classifications in the training set.
nb = MultinomialNB()
trainRes=[]
for r in posreviews:
    trainRes.append(1)
for r in negreviews:
    trainRes.append(-1)
nb.fit(train_features,trainRes)

# Now we can use the model to predict classifications for our test features.
predictions = nb.predict(test_features)
#print predictions
# Compute the error.  It is slightly different from our model because the internals of this process work differently from our imp
fpr, tpr, thresholds = metrics.roc_curve(actual, predictions, pos_label=1)
print("Multinomial naive bayes AUC: {0}".format(metrics.auc(fpr, tpr)))
```

```
Multinomial naive bayes AUC: 0.785085065614
```

Classifier Comparison

Choosing the right estimator

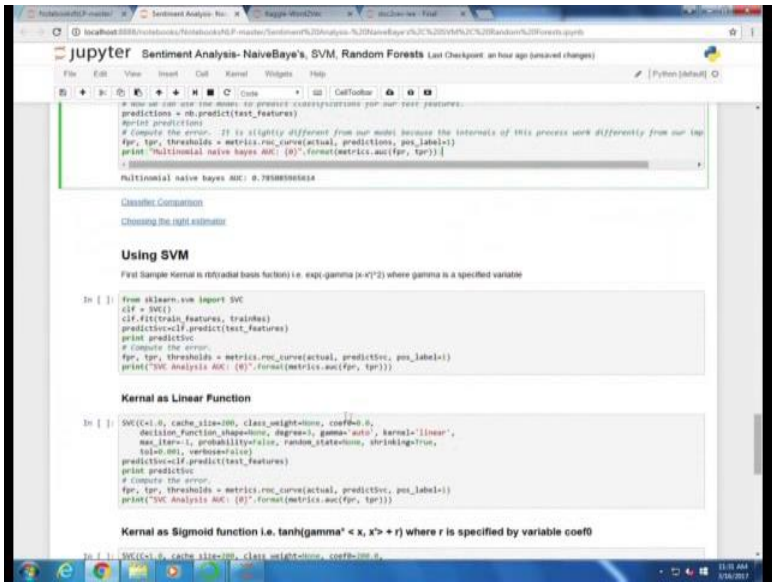
Using SVM

First Sample Kernal is rbf(radial basis fuction) i.e. exp(-gamma |x-x'|^2) where gamma is a specified variable

And then we convert each document to fit into any of to that vocabulary space in test features if a new word is appearing or that is called as out of vocabulary word that is ignored and only those counts are retained which are originally there in the vocabulary.

Once we have that we run this classifier and we also calculate the area under curve that gives us that basically give tells us about the false positive weight and the true positive weight, so for a binary classifier like this information helps us in identifying the performance of the classifier.

(Refer Slide Time: 18:14)



Sentiment Analysis- NaiveBaye's, SVM, Random Forests

```
predictions = nb.predict(test_features)
#print predictions
# Compute the error.  It is slightly different from our model because the internals of this process work differently from our imp
fpr, tpr, thresholds = metrics.roc_curve(actual, predictions, pos_label=1)
print "Multinomial naive bayes AUC: {0}".format(metrics.auc(fpr, tpr))
```

```
Multinomial naive bayes AUC: 0.785085965614
```

Classifier Comparison

Choosing the right estimator

Using SVM

First Sample Kernal is rbf(radial basis fuction) i.e. exp(-gamma |x-x'|^2) where gamma is a specified variable

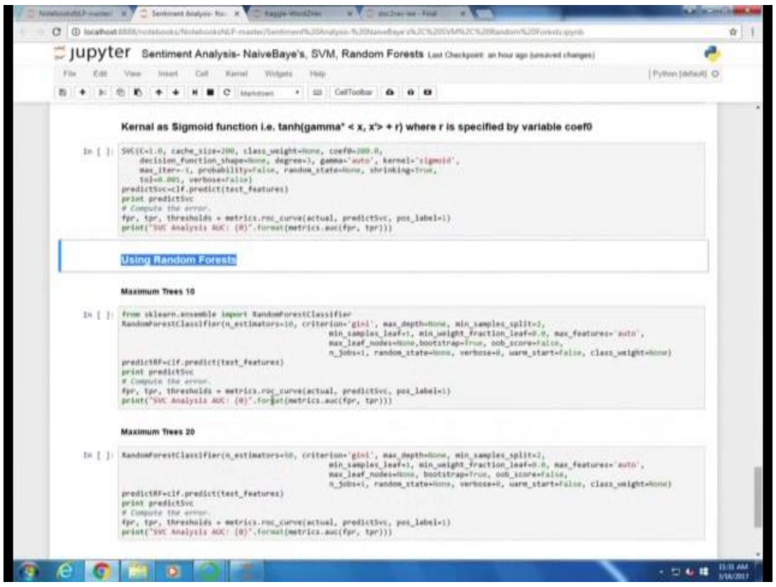
```
from sklearn.svm import SVC
clf = SVC()
clf.fit(train_features, trainRes)
predictSvc=clf.predict(test_features)
print predictSvc
# Compute the error.
fpr, tpr, thresholds = metrics.roc_curve(actual, predictSvc, pos_label=1)
print("SVC Analysis AUC: {0}".format(metrics.auc(fpr, tpr)))
```

Kernal as Linear Function

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
predictSvc=clf.predict(test_features)
print predictSvc
# Compute the error.
fpr, tpr, thresholds = metrics.roc_curve(actual, predictSvc, pos_label=1)
print("SVC Analysis AUC: {0}".format(metrics.auc(fpr, tpr)))
```

Kernal as Sigmoid function i.e. tanh(gamma* < x, x'> + r) where r is specified by variable coef0

(Refer Slide Time: 18:25)



Sentiment Analysis- NaiveBaye's, SVM, Random Forests

Kernal as Sigmoid function i.e. tanh(gamma* < x, x'> + r) where r is specified by variable coef0

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=200.0,
    decision_function_shape=None, degree=3, gamma='auto', kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
predictSvc=clf.predict(test_features)
print predictSvc
# Compute the error.
fpr, tpr, thresholds = metrics.roc_curve(actual, predictSvc, pos_label=1)
print("SVC Analysis AUC: {0}".format(metrics.auc(fpr, tpr)))
```

Using Random Forests

Maximum Trees 10

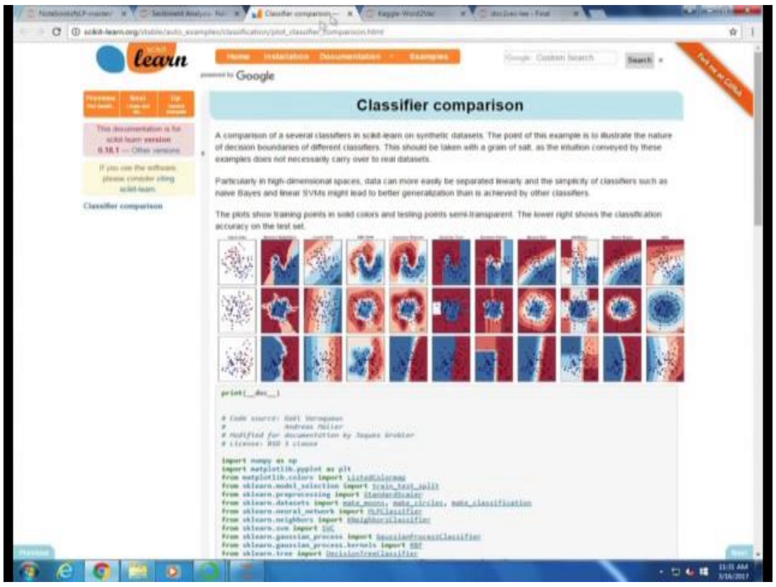
```
from sklearn.ensemble import RandomForestClassifier
RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
                                           min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
                                           max_leaf_nodes=None,bootstrap=True, oob_score=False,
                                           n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None)
predictRF=clf.predict(test_features)
print predictSvc
# Compute the error.
fpr, tpr, thresholds = metrics.roc_curve(actual, predictSvc, pos_label=1)
print("SVC Analysis AUC: {0}".format(metrics.auc(fpr, tpr)))
```

Maximum Trees 20

```
RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
                                           min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
                                           max_leaf_nodes=None, bootstrap=True, oob_score=False,
                                           n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None)
predictRF=clf.predict(test_features)
print predictSvc
# Compute the error.
fpr, tpr, thresholds = metrics.roc_curve(actual, predictSvc, pos_label=1)
print("SVC Analysis AUC: {0}".format(metrics.auc(fpr, tpr)))
```

So, Scikit learn provides a host of classifiers it includes classifiers like SVM with multiple Kernal options or a simple classifiers like random forests. So, you can refer to any of the machine learning lectures to have an understanding of these classifiers for a quick check of what are the different classifiers that is provided.

(Refer Slide Time: 18:40)



Classifier comparison

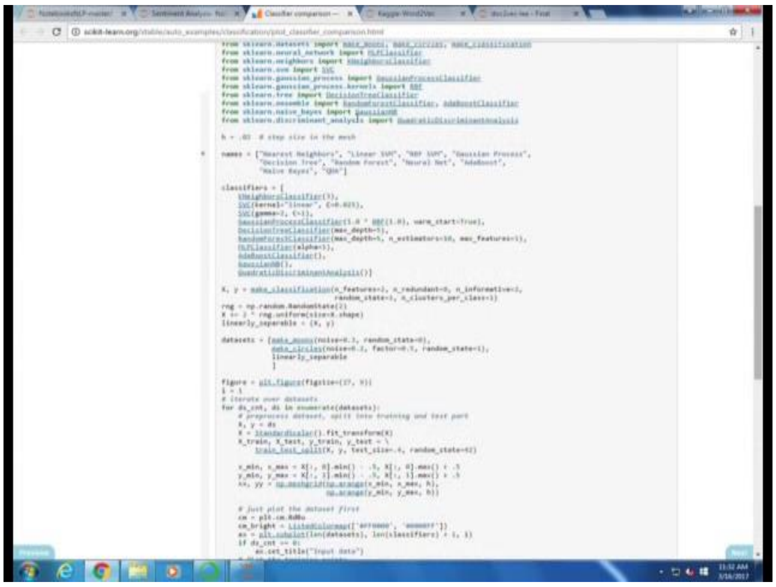
A comparison of a several classifiers in scikit-learn on synthetic datasets. The point of this example is to illustrate the nature of decision boundaries of different classifiers. This should be taken with a grain of salt, as the intuition conveyed by these examples does not necessarily carry over to real datasets.

Particularly in high-dimensional spaces, data can more easily be separated linearly and the simplicity of classifiers such as naive Bayes and linear SVMs might lead to better generalization than is achieved by other classifiers.

The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set.

```
print(__doc__)


# Code source: Gaël Varoquaux
#              Andreas Müller
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
```

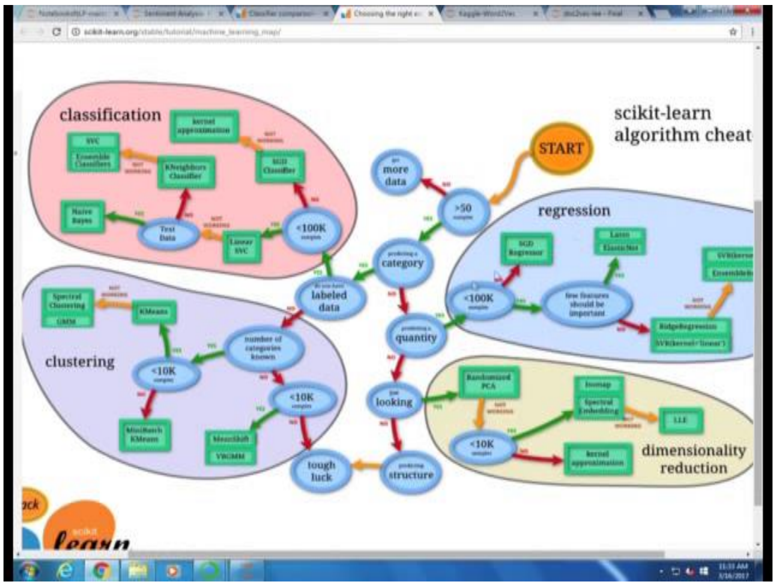
(Refer Slide Time: 18:40)



```
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

h = .02  # step size in the mesh

names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost",
         "Naive Bayes", "QDA"]

classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0), warm_start=True),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    MLPClassifier(alpha=1),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis()]

X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
                           random_state=1, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_separable = (X, y)

datasets = [make_moons(noise=0.3, random_state=0),
            make_circles(noise=0.2, factor=0.5, random_state=1),
            linearly_separable
            ]

figure = plt.figure(figsize=(27, 9))
i = 1
# iterate over datasets
for ds_cnt, ds in enumerate(datasets):
    # preprocess dataset, split into training and test part
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.4, random_state=42)

    x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
    y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    # just plot the dataset first
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    ax = plt.subplot(len(datasets), len(classifiers) + 1, i)
    if ds_cnt == 0:
        ax.set_title("Input data")
```

You can have a look into the scikit learns official documentation. So, the link is already provided here. So, here the classifier essentially provides different classifiers and they show on one data set how each of this classifier is performing you can have a look into this in addition.

(Refer Slide Time: 19:09)

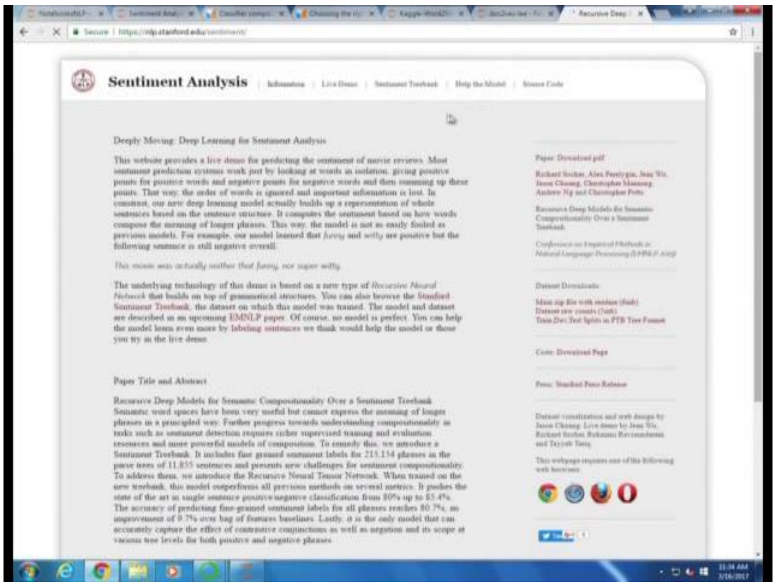


You can have a look into this quick scikit that will help you to find the relevant classifier or the estimator that you should be using. So, this scikit essentially provides all possible or it basically summarizes what are the different estimators that is available in scikit and you can see that when we do a supervise approach we should be either doing a classification or when we have where we have to classify or we have to predict a particular label.

In this case we were doing a classification where we predicting that whether something belongs to positive or negative. Suppose you wanted to predict the degree of positiveness or negativeness in a particular document you might have to model the same task as a regression task where it has a suite of library or suite of methods available here. In case of working with un supervised data like plastering or those kind of things that we have done in words disambiguation you might have to consider one of these clustering options or you can use the topic model that we have discussed in the previous tutorial.

This essentially summarizes the class the supervised approaches for building a sentiment analysis tool.

(Refer Slide Time: 21:00)



Deeply Moving: Deep Learning for Sentiment Analysis

This website provides a live demo for predicting the sentiment of movie reviews. Most sentiment prediction systems work just by looking at words in isolation, giving positive points for positive words and negative points for negative words and then summing up these points. That way, the order of words is ignored and important information is lost. In constrast, our new deep learning model actually builds up a representation of whole sentences based on the sentence structure. It computes the sentiment based on how words compose the meaning of longer phrases. This way, the model is not as easily fooled as previous models. For example, our model learned that *funny* and *witty* are positive but the following sentence is still negative overall:

This movie was actually neither that funny, nor super witty.

The underlying technology of this demo is based on a new type of *Recursive Neural Network* that builds on top of grammatical structures. You can also browse the Stanford Sentiment Treebank, the dataset on which this model was trained. The model and dataset are described in an upcoming EMNLP paper. Of course, no model is perfect. You can help the model learn even more by labeling sentences we think would help the model or those you try in the live demo.

Paper Title and Abstract

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank
Semantic word spaces have been very useful but cannot express the meaning of longer phrases in a principled way. Further progress towards understanding compositionality in tasks such as sentiment detection requires richer supervised training and evaluation resources and more powerful models of composition. To remedy this, we introduce a Sentiment Treebank. It includes fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences and presents new challenges for sentiment compositionality. To address them, we introduce the Recursive Neural Tensor Network. When trained on the new treebank, this model outperforms all previous methods on several metrics. It pushes the state of the art in single sentence positive/negative classification from 80% up to 85.4%. The accuracy of predicting fine-grained sentiment labels for all phrases reaches 80.7%, an improvement of 9.7% over bag of features baselines. Lastly, it is the only model that can accurately capture the effect of contrastive conjunctions as well as negation and its scope at various tree levels for both positive and negative phrases.

Paper: Download pdf

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng and Christopher Potts

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)

Dataset Downloads:

Main zip file with readme (6mb)
Dataset raw counts (5mb)
Train,Dev,Test Splits in PTB Tree Format

Code: Download Page

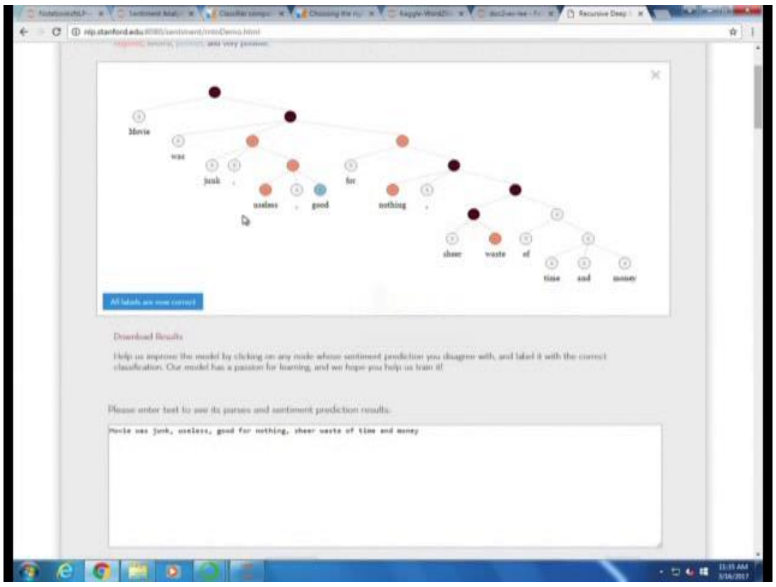
Press: Stanford Press Release

Dataset visualization and web design by Jason Chuang. Live demo by Jean Wu, Richard Socher, Rukmani Ravisundaram and Tayyab Tariq.

This webpage requires one of the following web browsers:

You can also look into other standard sentiment analysis tools like the Stanford sentiment analyzer see if we give one of our sentences that we have used here for testing.

(Refer Slide Time: 21:37)



Download Results

Help us improve the model by clicking on any node whose sentiment prediction you disagree with, and label it with the correct classification. Our model has a passion for learning, and we hope you help us train it!

Please enter text to see its parses and sentiment prediction results:

```
Movie was junk, useless, good for nothing, sheer waste of time and money
```

Here what we can find is that this again shows more or less negative sentiments to the classic the sentence. So, it basically shows each word to be either very negative or negative and very few words to have a positive impact. So, that will be all for this tutorial, please try out other classifiers as well.

Thank you.